Contents

Searching in Arrays

We now discuss how we can search for values in an array.

Finding the Maximum Value

To find the greatest value in an array of integer, one needs a comparison point, a variable holding "the greatest value *so far*". Once this value is set, then one "just" have to inspect each value in the array, and to update "the greatest value *so far*" if the value currently inspected is greater, and then to move on to the next value. Once we reach the end of the array, we know that "the greatest value *so far*" is actually the greatest value (period) in the array.

The problem is to find the starting point: one cannot assume that "the greatest value *so far*" is 0 (what if the array contains only negative values?), so the best strategy is simply to assume that "the greatest value *so far*" is the first one in the array (after all, it *is* the greatest value we have seen *so far*).

Using **`foreach`**, we have for example the following:

```
int[] arrayExample = { 1, 8, -12, 9, 10, 1, 30, 1, 32, 3
 ↪  };

int maxSoFar = arrayExample[0];
foreach (int i in arrayExample)
    if (i > maxSoFar) maxSoFar = i;

Console.WriteLine("The greatest value is "
    + maxSoFar + ".");
```

Finding a Particular Value

Suppose we want to set a particular Boolean variable to **`true`** if a particular value `target` is present in an arrayy `arrayExample`. The simplest

way to perform such a search is to

1. Set the Boolean variable to **false**,
2. Inspect the values in `arrayExample` one by one, comparing them to `target`, and setting the Boolean variable to **true** if they are identical.

```
int[] arrayExample = { 1, 8, -12, 9, 10, 1, 30, 1, 32, 3
 ↪  };

bool foundTarget = false;
int target = 8;

for (int i = 0; i < arrayExample.Length; i++)
{
    if (arrayExample[i] == target) foundTarget = true;
}
Console.WriteLine(target + " is in the array: " +
 ↪  foundTarget + ".");
```

Note that in the particular example above, we could have stopped exploring the array after the second index, since the target value was found. A slightly different logic would allow to exit prematurely the loop when the `target` value is found:

```
int[] arrayExample = { 1, 8, -12, 9, 10, 1, 30, 1, 32, 3
 ↪  };

bool foundYet = false;
int target = 30;
int index = 0;

do
{
    if (arrayExample[index] == target) foundYet = true;
    index++;
}
while (index < arrayExample.Length && !foundYet);
Console.WriteLine(target + " is in the array: " +
 ↪  foundYet +
"\nNumber of elements inspected: " + (index) +".");
```

This code would display:

```
30 is in the array: True
Number of elements inspected: 7.
```

Both codes are examples of *linear* (or *sequential*) *search*: the array is parsed one element after the other, and potentially all elements are in-

spected.

Finding a Particular Value in a Sorted Array

If the array is *sorted* (that is, the value at index i is less than the value at index $i + 1$), then the search for a particular value can be sped up by using *binary search*.

Sorted Arrays

A way of making sure that an array is sorted is given below. Note that, as above when trying to find the maximum value, we decide that the array is "sorted so far" unless proven otherwise, in which case we exit prematurely the loop. Note also that the condition contains `index + 1 < arrayExample.Length`: we need to make sure that "the next value" actually exists before comparing it with the current value.

```
int[] arrayExample = { 1, 10, 12, -1};
bool sortedSoFar = true;
int index = 0;

while (index + 1 < arrayExample.Length && sortedSoFar)
{
    if (arrayExample[index] > arrayExample[index+1])
     ↪  sortedSoFar = false;
    index++;
}
Console.WriteLine("The array is sorted: " + sortedSoFar
 ↪  +".");
```

Binary Search

Introduction *Binary* (*half-interval* or *logarithmic*) *search* leverages the fact that the array is sorted to speed up the search for a particular value. It goes as follows:

The algorithm compares the `target` value to the middle element of the array.

1. If they are equal, we are done.
2. If they are not equal, then there are two cases:
   (a) If the middle element is greater than the `target`, then the algorithm restarts, but looking for the value only in the left half of the array,

(b) If the middle element is less than the `target`, then the algorithm restarts, but looking for the value only in the right half of the array.

3. If the search ends with the remaining half being empty, the `target` is not in the array.

First Example An example of implementation (and of execution) is as follows:

```
int[] arrayExample = { 1, 10, 12, 129, 190, 220, 230,
 ↪  310, 320, 340, 400, 460};
bool foundSoFar = false;

int target = 340;

int start = 0;
int end = arrayExample.Length - 1;
int mid;
while (start <= end && !foundSoFar)
{
    mid = (start + end) / 2;
    /*
     * This is integer division: if start + end is odd,
     * then it will be truncated. In our example,
     * (0 + 11) / 2 gives 5.
     */
    Console.WriteLine("The middle index is " + mid + ".");
    if (target == arrayExample[mid])
    {
        foundSoFar = true;
    }
    else if (target > arrayExample[mid])
    {
        start = mid + 1;
        Console.WriteLine("I keep looking right.");
    }
    else
    {
        end = mid - 1;
        Console.WriteLine("I keep looking left.");
    }
}
Console.WriteLine("Found the value: " + foundSoFar +".");
```

This code would display:

```
The middle index is 5.
I keep looking right.
The middle index is 8.
I keep looking right.
The middle index is 10.
I keep looking left.
The middle index is 9.
Found the value: True.
```

Second Example Remembering that characters are such that `'A'` is less than `'a'`, and `'a'` is less than `'b'`, we can run a binary search on a sorted array of characters. The code below is the same algorithm as above, only the information logged changes:

```
char[] arrayExample = { 'A', 'B', 'D', 'Z', 'a', 'b', 'd'
 ↪  };
char target = 'D';
bool foundSoFar = false;
int start = 0;
int end = arrayExample.Length - 1;
int mid;
while (start <= end && !foundSoFar)
{
    Console.WriteLine("Range: " + start + " -- " + end);
    mid = (start + end) / 2;
    Console.WriteLine("Mid: " + mid);
    if (target == arrayExample[mid])
    {
        foundSoFar = true;
    }
    else if (target > arrayExample[mid])
    {
        start = mid + 1;
    }
    else
    {
        end = mid - 1;
    }
}
Console.WriteLine("Found the value: " + foundSoFar + ".");
```

This will display:

```
Range: 0 -- 6
Mid: 3
Range: 0 -- 2
Mid: 1
```

```
Range: 2 -- 2
Mid: 2
Found the value: True
```

Observe that if we were to replace `start <= end` with `start < end` then the algorithm would not have correctly terminated in the example above.